**QUIZ-ENGINE PROJECT**

This C++ program is a simple console quiz program developed based on the principles of object-oriented programming (OOP). It supports two types of users—GuestUser and PremiumUser—with different permissions when responding to the quiz. The program also illustrates the usage of dynamic memory allocation, pointer handling, inheritance, and abstraction.

1. **Including Required Libraries**

#include <iostream>  
#include <cstring>

The program starts by including two header files. iostream is used for input and output operations such as cin and cout. cstring provides functions for string manipulation (e.g., strncpy) for copying character arrays.

using namespace std;

This line tells the compiler to use the standard namespace, so we don’t have to prefix std:: every time we use cout, cin, or other standard objects.

1. **Defining the Question Structure**

struct Question {  
 char prompt[100];  
 char choices[4][40];  
 int correctIdx;  
};

Here, a structure Question is declared to represent a quiz question. It contains:

* prompt: a string to hold the question text.
* choices: a 2D array to store four possible answer options.
* correctIdx: an integer that stores the index of the correct answer (from 0 to 3).

1. **Declaring the Abstract Base Class: QuizUser**

class QuizUser {  
public:  
 virtual int attemptQuiz(Question\* questions, int count) = 0;  
 virtual ~QuizUser() {}  
};

This is an abstract base class called QuizUser containing:

* A pure virtual function attemptQuiz, which must be overridden by any derived class. It takes a pointer to an array of questions and a count of how many there are.
* A virtual destructor, ensuring proper cleanup of resources if a derived class object is deleted via a base class pointer.

1. **GuestUser Class Definition**

class GuestUser : public QuizUser {

This class publicly inherits from QuizUser, meaning it must implement attemptQuiz

int attemptQuiz(Question\* questions, int count) override {

The attemptQuiz method allows a Guest user to answer up to 3 questions

int limit = (count < 3) ? count : 3;.

If there are fewer than 3 questions, the limit is adjusted accordingly.

for (int i = 0; i < limit; i++) {  
 ...  
 if (answer - 1 == q->correctIdx) score++;  
}

The loop displays each question, allows the user to input an answer, and compares it to the correct index. The score is increased if the answer is correct.

1. **PremiumUser Class Definition**

class PremiumUser : public QuizUser {

A Premium user can attempt all available questions. The attemptQuiz method here is similar to GuestUser but uses the full count rather than a limit of 3.

1. **Adding Questions Dynamically**

void addQuestion(Question\*& questions, int& size, Question newQ)

This function adds a new Question to a dynamically allocated array:

* It creates a new array one element larger.
* Copies all old questions into the new array.
* Adds the new question at the end.
* Deletes the old array and updates the pointer and size.

1. **Removing a Question**

void removeQuestion(Question\*& questions, int& size, int index)

This function removes a question at a specified index. It:  
Validates the index.  
Creates a new array one element smaller.  
Copies all questions except the one to remove.  
Deletes the old array and updates the size.

1. **Creating Questions Manually**

Question createQuestion(...)

This helper function simplifies question creation. It:

* Copies the prompt and choices using strncpy for safety.
* Sets the correct answer index.

Returns a Question object.

**9. Main Function: Execution Starts Here**

int main()

The main function orchestrates the entire flow:

* Initializes a dynamic questions array with 0 size.
* Adds four sample questions using addQuestion and createQuestion.

QuizUser\*\* users = new QuizUser\*[userCount];

An array of QuizUser pointers is created. The first is a GuestUser, and the second a PremiumUser

for (int i = 0; i < userCount; i++) {  
 users[i]->attemptQuiz(questions, qSize);  
}.

Each user takes the quiz according to their type. The virtual function ensures the correct version of attemptQuiz is called, thanks to polymorphism.

**10. Cleaning Up Memory.**

for (int i = 0; i < userCount; i++) {  
 delete users[i];  
}  
delete[] users;  
delete[] questions;

Memory allocated using new must be manually freed using delete. This prevents memory leaks, which is especially important in larger programs.

Conclusion  
This quiz program is a good example of proper usage of object-oriented programming principles, especially abstraction, inheritance, and polymorphism. The program also deals with dynamic memory using explicit allocation and deallocation. The Guest/Premium user type difference introduces real-world constraints and shows how the behavior of objects may vary with respect to class hierarchy. Overall, it's a well-structured and neat C++ implementation of a minimalist but flexible quiz system.